TWO NEW WAYS TO EXPLOIT A FIXED BROWSER FINGERPRINTING FLAW

XIAOYIN LIU

BSIDES MUNICH 2019

MARCH 25, 2019

ABOUT ME

- Independent bug bounty hunter
- Graduated from University of North Carolina at Chapel Hill, United States
- Recognized by Microsoft, Google, Brave Browser, Tor Project and more
- Areas of interest: Windows applications, web browsers

OUTLINE

- Browser fingerprinting issues
- Background of a fingerprinting flaw, Sniffly
- First bypass
- Second bypass
- Takeaways
- CVE-2017-0135

BROWSER FINGERPRINTING ISSUES

- Fingerprinting is privacy issues
- Some browser vendors, like Tor Browser, are more interested in fixing fingerprinting issues than Chrome, Firefox, Edge, etc.
- Examples: HSTS super cookie, CSS Visited, etc.

SNIFFLY ATTACK

- Discovered by Yan Zhu in 2015
- Abusing HSTS/301 redirect and CSP to probe user’s browsing history
- CVE-2016-1617

HTTP STRICT TRANSPORT SECURITY (HSTS)

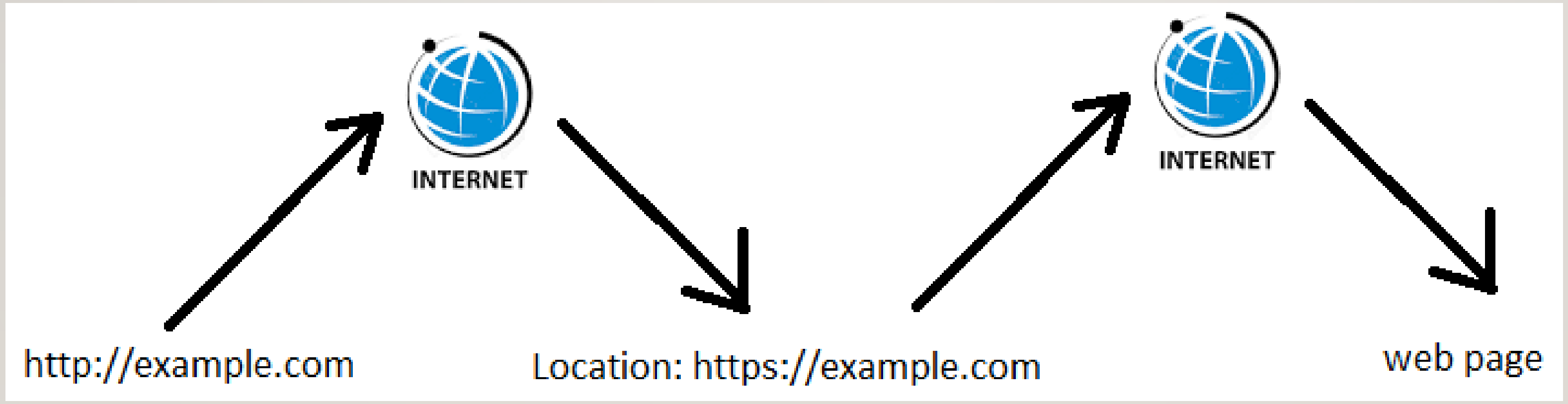
- A browser security feature that enforces HTTPS for all connections for particular domains
- Strict-Transport-Security: max-age=604800

CONTENT SECURITY POLICY (CSP)

- Security feature to mitigate XSS attacks
- Content-Security-Policy: script-src ‘self’ www.google.com; img-src ‘self’; default-src ‘none’
- CSP is also used to enable other security features, like Upgrade Insecure Requests.

HOW SNIFFLY WORKS

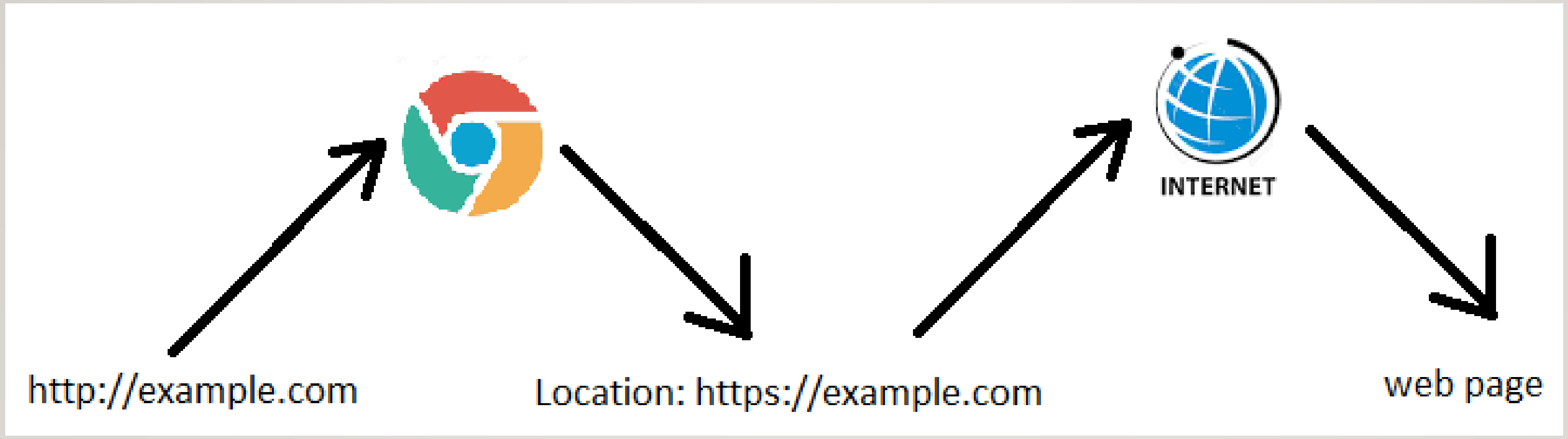
- Attacker embeds an image tag <img src=http://example.com>
- Attacker knows that example.com is either a HSTS domain or 301 redirects to HTTPS
- If a visitor has never visited http://example.com:



https://github.com/diracdeltas/sniffly

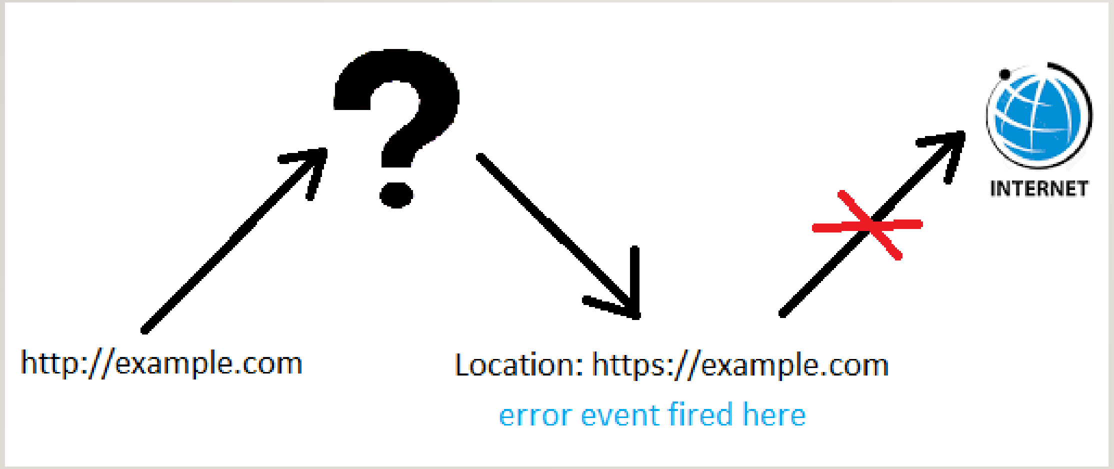
HOW SNIFFLY WORKS

- If a visitor has visited http://example.com:



HOW SNIFFLY WORKS

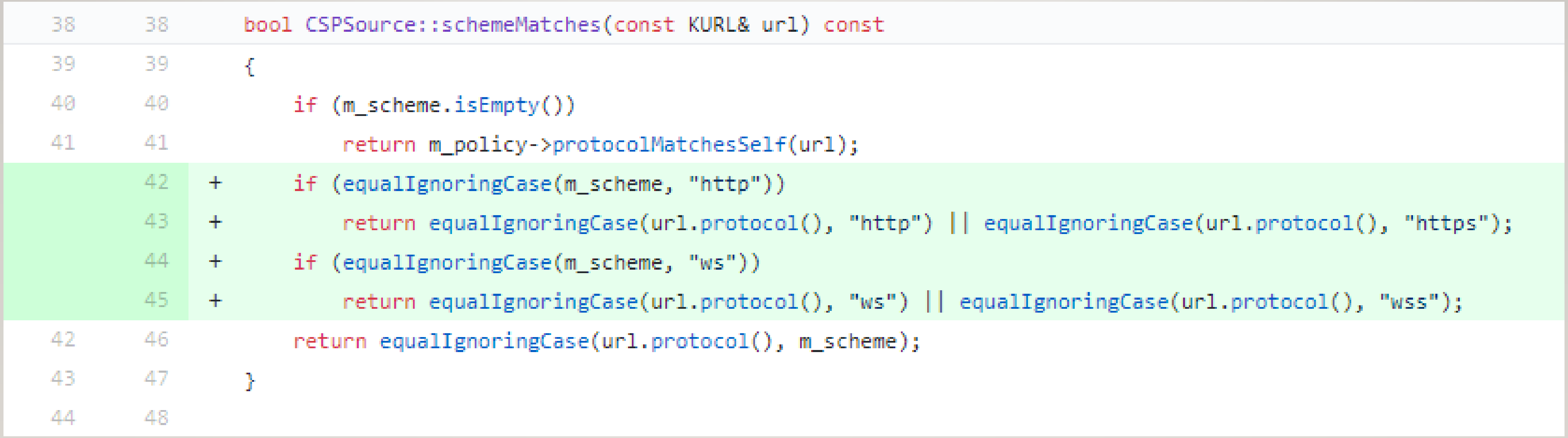
- If we can time how long it takes to receive the redirect response, we can distinguish if it's an in-browser redirect or a network redirect.
- Use CSP to allow http requests but block https requests:
- Content-Security-Policy: img-src http://example.com



HOW SNIFFLY WORKS

- If the error event is triggered within a threshold (10ms), it’s an internal redirect. Then this URL has been visited before; otherwise, it has not been visited.

THE CODE SNIPPET



```
 bool CSPSource::schemeMatches(const KURL& url) const
 {
     if (m_scheme.isEmpty())
         return m_policy->protocolMatchesSelf(url);
+    if (equalIgnoringCase(m_scheme, "http"))
+        return equalIgnoringCase(url.protocol(), "http") || equalIgnoringCase(url.protocol(), "https");
+    if (equalIgnoringCase(m_scheme, "ws"))
+        return equalIgnoringCase(url.protocol(), "ws") || equalIgnoringCase(url.protocol(), "wss");
     return equalIgnoringCase(url.protocol(), m_scheme);
 }
```

https://github.com/chromium/chromium/commit/568075bbc5d16239a5cbdeb579a8768f9836f13e

- Content-Security-Policy: img-src http://example.com

  now matches both http://example.com and https://example.com

BYPASSING THE FIX

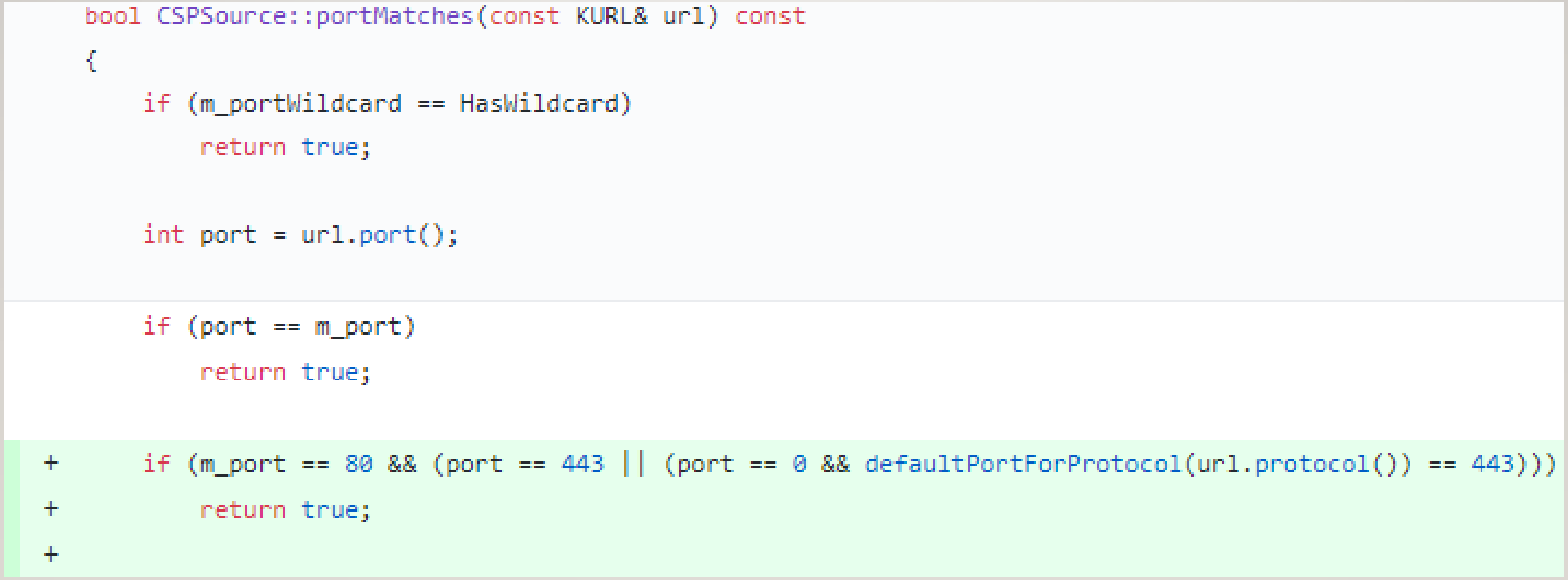
- The code only considers the protocol, not port
- Consider this CSP rule:

  Content-Security-Policy: img-src http://example.com:80

- Does it match https://example.com

- It turns out this CSP matches http://example.com:80 and https://example.com:80
- It doesn’t match https://example.com:443
- So we can exploit Sniffly again!
- CVE-2016-5137: https://bugs.chromium.org/p/chromium/issues/detail?id=625945
- CVE-2016-9071: https://bugzilla.mozilla.org/show_bug.cgi?id=1285003
- $1000 bounty

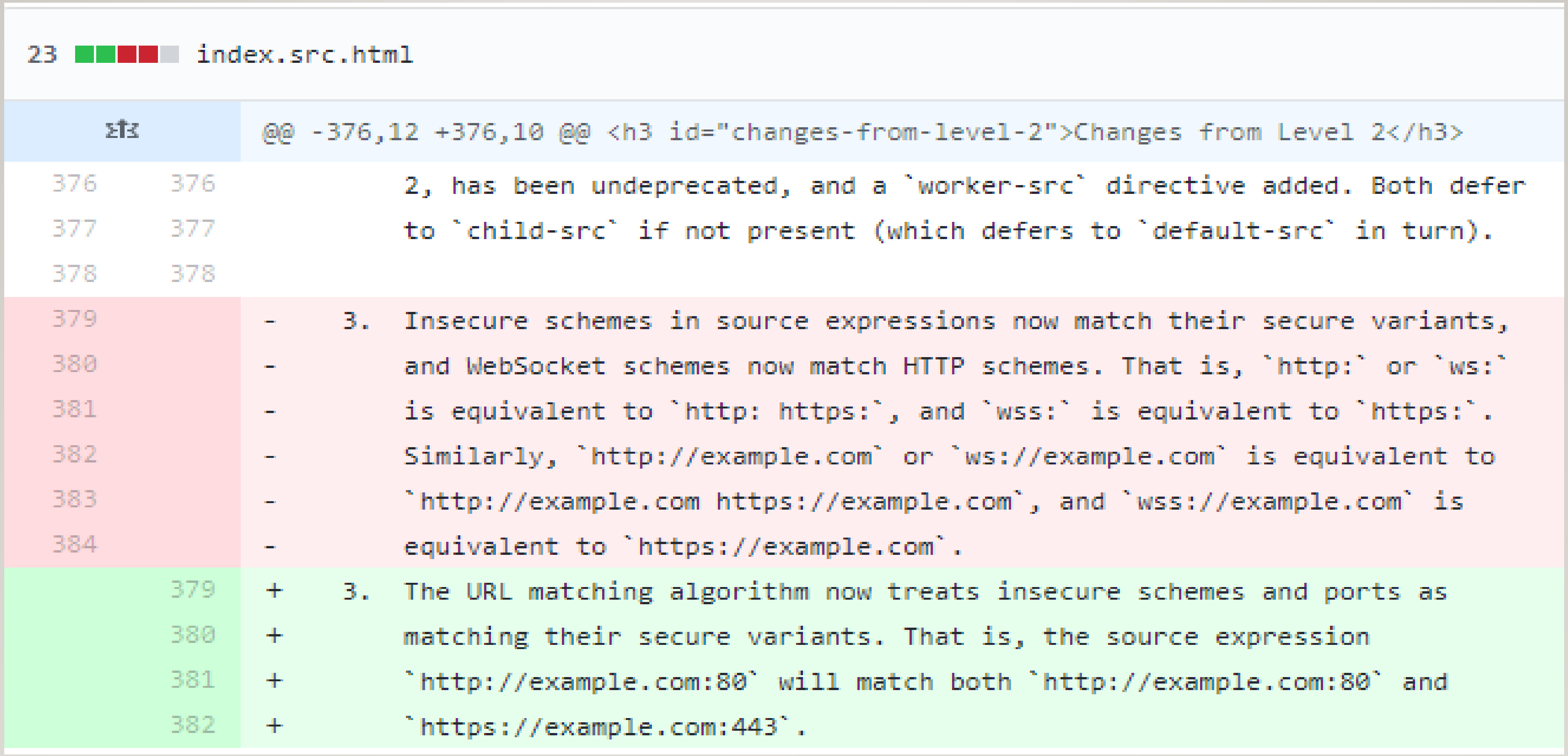
PATCH



```
    bool CSPSource::portMatches(const KURL& url) const
    {
        if (m_portWildcard == HasWildcard)
            return true;

        int port = url.port();

        if (port == m_port)
            return true;

+       if (m_port == 80 && (port == 443 || (port == 0 && defaultPortForProtocol(url.protocol()) == 443)))
+           return true;
+
```

https://github.com/chromium/chromium/commit/e6d181417ea462ac221d768c960a21018266a4a8

CHANGE IN CSP SPECIFICATION



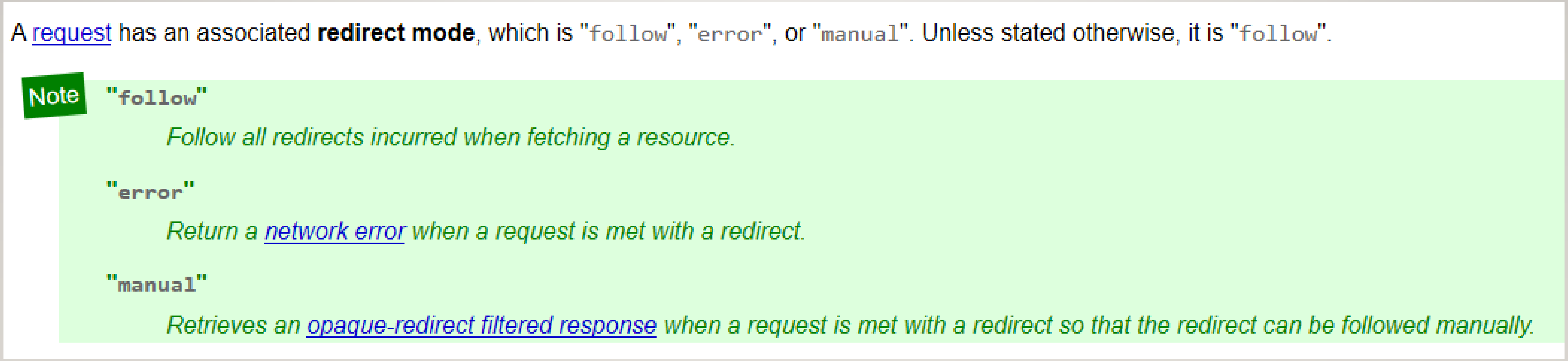
```
23 🟩🟩🟥🟥⬜ index.src.html

@@ -376,12 +376,10 @@ <h3 id="changes-from-level-2">Changes from Level 2</h3>
              2, has been undeprecated, and a `worker-src` directive added. Both defer
              to `child-src` if not present (which defers to `default-src` in turn).

-       3.    Insecure schemes in source expressions now match their secure variants,
-             and WebSocket schemes now match HTTP schemes. That is, `http:` or `ws:`
-             is equivalent to `http: https:`, and `wss:` is equivalent to `https:`.
-             Similarly, `http://example.com` or `ws://example.com` is equivalent to
-             `http://example.com https://example.com`, and `wss://example.com` is
-             equivalent to `https://example.com`.
+       3.    The URL matching algorithm now treats insecure schemes and ports as
+             matching their secure variants. That is, the source expression
+             `http://example.com:80` will match both `http://example.com:80` and
+             `https://example.com:443`.
```

https://github.com/w3c/webappsec-csp/commit/22d08b990290e49f5a666fad08de16d75bb369e7

SECOND BYPASS

- So far both attacks use CSP to block the redirect
- Are there other ways to achieve the same?
- Use Fetch API

FETCH API



A request has an associated **redirect mode**, which is "`follow`", "`error`", or "`manual`". Unless stated otherwise, it is "`follow`".

Note

"`follow`"

Follow all redirects incurred when fetching a resource.

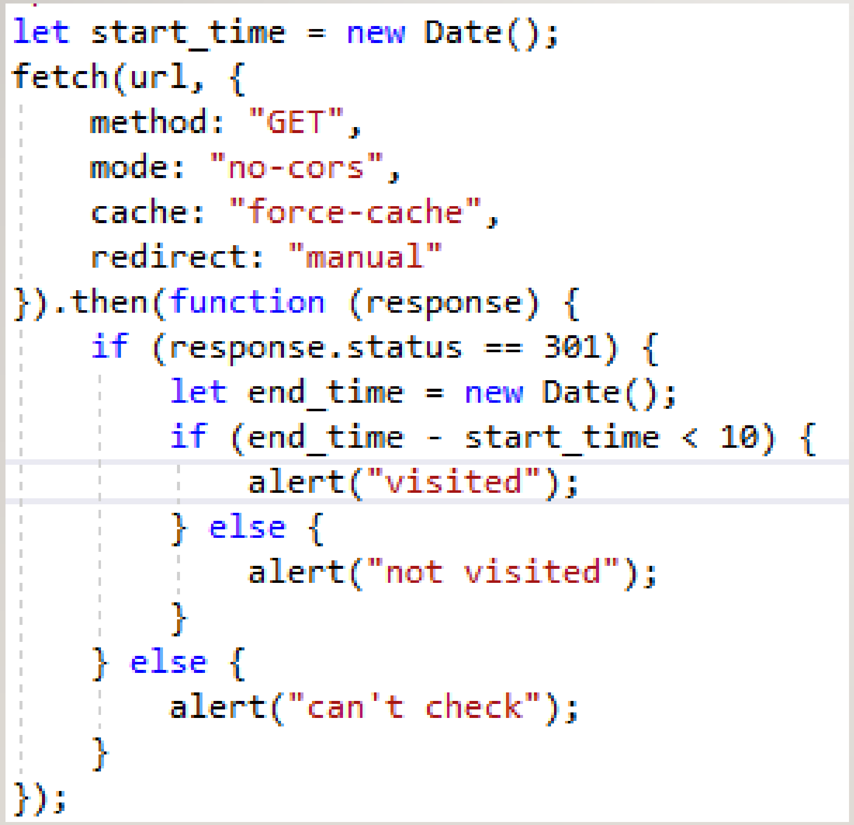
"`error`"

Return a network error when a request is met with a redirect.

"`manual`"

Retrieves an opaque-redirect filtered response when a request is met with a redirect so that the redirect can be followed manually.

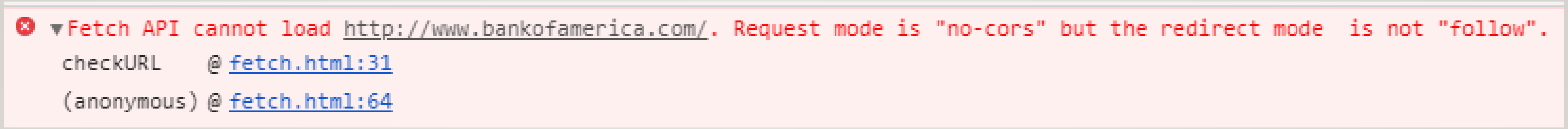
FETCH API



```
let start_time = new Date();
fetch(url, {
    method: "GET",
    mode: "no-cors",
    cache: "force-cache",
    redirect: "manual"
}).then(function (response) {
    if (response.status == 301) {
        let end_time = new Date();
        if (end_time - start_time < 10) {
            alert("visited");
        } else {
            alert("not visited");
        }
    } else {
        alert("can't check");
    }
});
```

FETCH API

- Reported in 2016. Not updated for more than 2 years.
- Silently fixed recently.
- In current Chrome, "no-cors" can't be used together with "manual" redirect



```
▼Fetch API cannot load http://www.bankofamerica.com/. Request mode is "no-cors" but the redirect mode  is not "follow".
  checkURL    @ fetch.html:31
  (anonymous) @ fetch.html:64
```

TAKEAWAYS

- Reading disclosed vulnerability reports and the patches is helpful for finding new ones
- Try to find corner cases that developers may neglect to handle (e.g. explicit port in URL)
- Mainstream browser vendors are generally not interested in fixing fingerprint issues

ANOTHER VULNERABILITY

- This is not a fingerprinting issue. It is an example to show how I find a real vulnerability by reading bug reports.
- CVE-2017-0135

CVE-2017-0135

- Inspired by paper "Abusing Internet Explorer 8's XSS Filters", written by Eduardo Vela Nava and David Lindsay
- How IE XSS Filter works: it checks if any URL parameter seems to be a XSS payload and then checks if the parameter is contained in HTML response
- http://example.com/index.php?id=<script>alert(1)</script>
- If HTML body contains <script>alert(1)</script>, then it's changed to <sc#ipt>alert(1)</script>

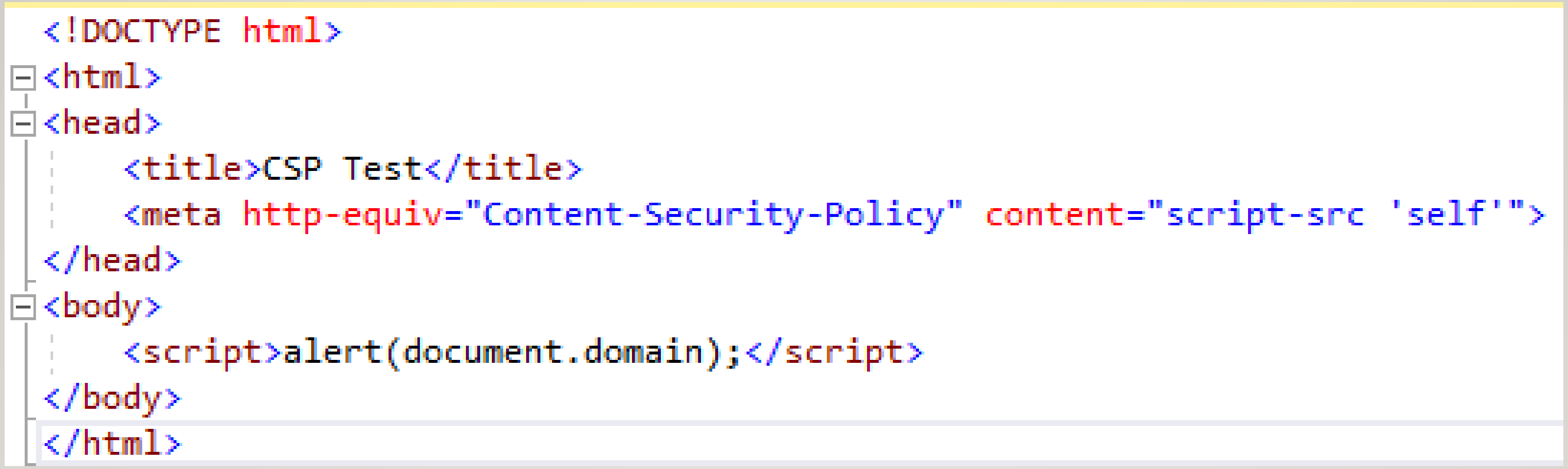
CVE-2017-0135

- What if it’s not a reflected XSS, but an expected JS code
- E.g. example.com/index.php?<script src="jquery.js"></script>
- <sc#ipt src="jquery.js"></script>
- Then jquery.js won’t load
- Seems harmless

CVE-2017-0135

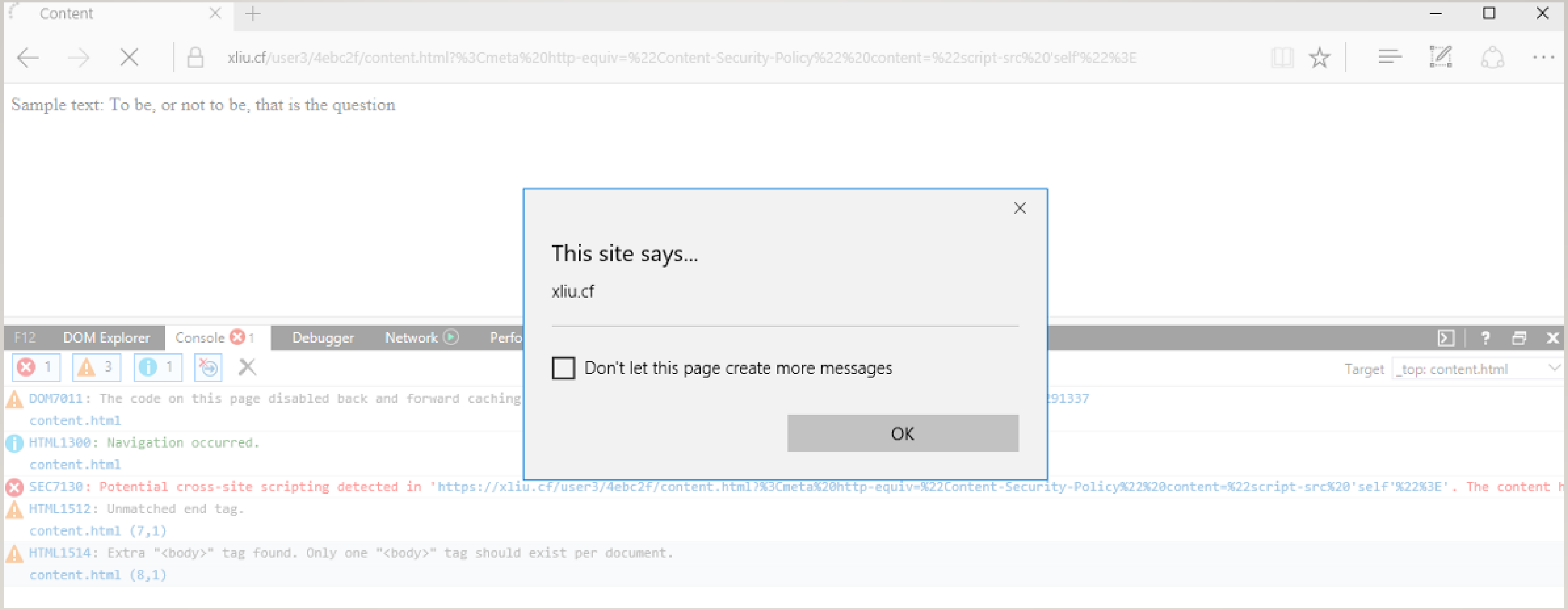
- Abuse XSS Filter to disable CSP
- <meta http-equiv="Content-Security-Policy" content="script-src 'self'">
- example.com/xss.html?<meta http-equiv="Content-Security-Policy" content="script-src 'self'">

CVE-2017-0135



```
<!DOCTYPE html>
<html>
<head>
    <title>CSP Test</title>
    <meta http-equiv="Content-Security-Policy" content="script-src 'self'">
</head>
<body>
    <script>alert(document.domain);</script>
</body>
</html>
```

CVE-2017-0135


Content
xliu.cf/user3/4ebc2f/content.html?%3Cmeta%20http-equiv=%22Content-Security-Policy%22%20content=%22script-src%20'self'%22%3E
F12
DOM Explorer
Console 1
Debugger
Network
Perfo
Target _top: content.html

Sample text: To be, or not to be, that is the question

This site says...

xliu.cf

Don't let this page create more messages

OK

```
DOM7011: The code on this page disabled back and forward caching
content.html
HTML1300: Navigation occurred.
content.html
SEC7130: Potential cross-site scripting detected in 'https://xliu.cf/user3/4ebc2f/content.html?%3Cmeta%20http-equiv=%22Content-Security-Policy%22%20content=%22script-src%20'self'%22%3E'. The content h
HTML1512: Unmatched end tag.
content.html (7,1)
HTML1514: Extra "<body>" tag found. Only one "<body>" tag should exist per document.
content.html (8,1)
```

CVE-2017-0135

- Reported on December 2, 2016. Fixed on March 14, 2017.
- Bounty: $1500
- Microsoft removed XSS Filter in Edge in October 2018 Update

REFERENCES

- https://chromium.googlesource.com/chromium/src/+/master/docs/security/faq.md
- https://zyan.scripts.mit.edu/presentations/toorcon2015.pdf
- https://bugs.chromium.org/p/chromium/issues/detail?id=544765
- https://bugs.chromium.org/p/chromium/issues/detail?id=625945
- https://fetch.spec.whatwg.org/
- http://p42.us/ie8xss/Abusing_IE8s_XSS_Filters.pdf
- https://blogs.windows.com/windowsexperience/2018/07/25/announcing-windows-10-insider-preview-build-17723-and-build-18204

Q & A

- Thank you for your listening!